Recognition of Users' Activities using Constraint Satisfaction

Recognition of Users' Activities using Constraint Satisfaction

Swapna Reddy, Ya'akov Gal, and Stuart M. Shieber

School of Engineering and Applied Sciences
Harvard University

Abstract. Ideally designed software allow users to explore and pursue interleaving plans, making it challenging to automatically recognize user interactions. The recognition algorithms presented use constraint satisfaction techniques to compare user interaction histories to a set of ideal solutions. We evaluate these algorithms on data obtained from user interactions with a commercially available pedagogical software, and find that these algorithms identified users' activities with 93% accuracy.

1 Introduction

Computer systems often serve to aid human professionals and care-givers [1]. In these settings, a key requirement is the recognition of user activities, which is important for (1) informing care-givers about user performance, (2) facilitating machine-generated support, and (3) understanding how software is used.

Traditional plan recognition approaches assume agents who form a single, correct plan to achieve their goal. In contrast, flexible software allows users to experiment; users may make mistakes and pursue multiple, interleaving plans. Reasoning about every way that a user can interact in such systems is infeasible.

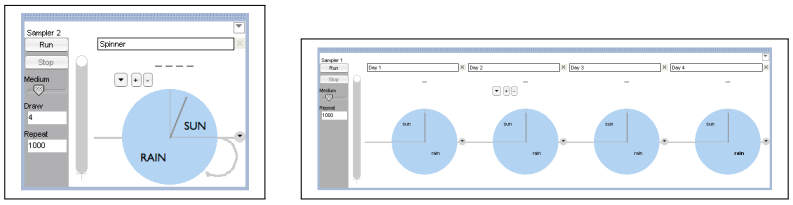
This paper presents two recognition algorithms for flexible software that use constraint satisfaction techniques to compare user interactions to ideal solutions designed by domain experts. We evaluate our algorithms empirically using a commercially-available pedagogical software, and we show that our methods outperform a recently proposed approach for inferring users' activities [2].

Many past recognition approaches query the user for clarification to reduce the search space of possible plans [3, 4]. However, interrupting users impedes their satisfaction and performance, and user replies cannot be assumed correct. Some non-intrusive approaches use machine learning to predict user intentions given past behavior [5, 6]. These assumptions do not hold in pedagogical domains, where students continuously solve new problems. Our work is also distinguished from probabilistic approaches that predict future user actions given recent interactions [7, 8]. We address a different problem, that of recognizing complete plans given entire interaction histories.

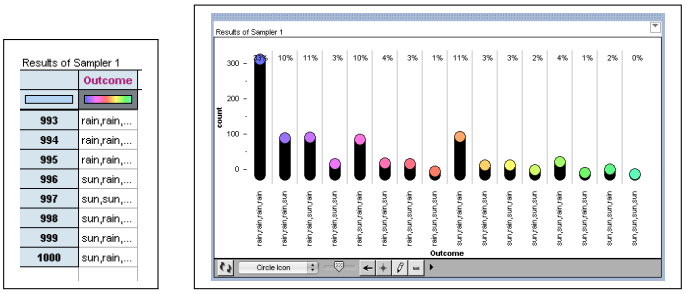
2 The TinkerPlots Domain

Our study involves TinkerPlots, a commercial software used worldwide to teach students in grades 4 through 8 about statistics and mathematics [9]. TinkerPlots is flexible, providing users with a "construction kit" for data to be modeled, generated, and analyzed in many ways using an open-ended interface [10].

Swapna Reddy, Ya'akov Gal, and Stuart M. Shieber. Recognition of users' activities using constraint satisfaction. In *Proceedings of the First and Seventeenth International Conference on User Modeling, Adaptation, and Personalization*, Trento, Italy, 22–26 June 2009.



(a) Two Possible Sampler Models



(b) Generating Sampler Data

Fig. 1: Solving the RAIN Problem with TinkerPlots

Our empirical studies focused on four problems for which students used TinkerPlots to estimate probabilities. As an example, we will use one of these problems, called RAIN: "The probability of rain on any day is 75%. Use TinkerPlots to compute the probability that it will rain on the next four consecutive days."

Two approaches to solving RAIN are shown in Figure 1. The first model in Figure 1a shows a sampler containing a "spinner" device with two possible events, "rain" and "sun". The distribution mass, or surface area, of the "rain" event is three times that of "sun". Each spin of this sampler samples the weather for one day. The number of spins is set to four, making the sampler a stochastic weather model for four consecutive days. A second model is shown in Figure 1b. This sampler contains four devices, each modeling a single day, and is spun once. Many other approaches are not shown. Figure 1b shows data as generated by a valid sampler and then organized into a histogram to infer likelihoods.

3 Actions, Recipes and Restrictions

Actions can be basic or complex. *Basic actions* are achieved directly, often with a single mouse or menu operation. TinkerPlots interactions are recorded as a sequence of basic actions, each with an ID and parametrized. Figure 2 shows a partial interaction for creating a device of Figure 1a, with parameters omitted for simplicity. Actions are abbreviated: ADS (Add Device to Sampler), AED (Add Event to Device), AS (Add Sampler), CPD (Change Probability in Device).

$$\ldots, \mathsf{ADS}, \mathsf{AED}, \mathsf{AS}, \mathsf{AED}, \mathsf{CPD}, \mathsf{ADS}, \ldots$$

Fig. 2: Partial Interaction History.

Complex actions are achieved indirectly through the completion of other actions, called *sub-actions*, which are themselves basic or complex actions [11]. Examples of complex actions include solving RAIN or fitting data to a plot.

A *recipe* for a complex action contains a set of sub-actions for achieving that complex action and a set of restrictions on those sub-actions [12]. Restrictions may limit the ordering of sub-actions or enforce relations among actions' parameters. We further require recipes to be non-recursive. Figure 3 provides one recipe for the complex action CCD (Create Correct Device), with constraints omitted for simplicity. This recipe includes the basic sub-actions ADS and CPD and the complex sub-action AE (Add Event).

$$\boxed{\text{CCD}} \longrightarrow \text{ADS}, \boxed{\text{AE}}, \boxed{\text{AE}}, \text{CPD}$$

Fig. 3: A Recipe for Creating a Device for the RAIN problem

An *expansion* of a complex action is a set of basic actions and restrictions that constitute completing that complex action. That is, an expansion is a recipe containing only basic sub-actions. Our recognition algorithms consist of two stages: generating all expansions for the desired complex action and comparing each expansion to the interaction history.

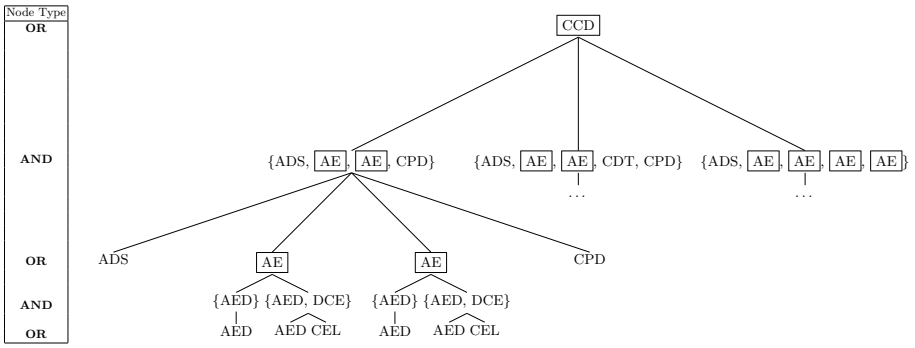
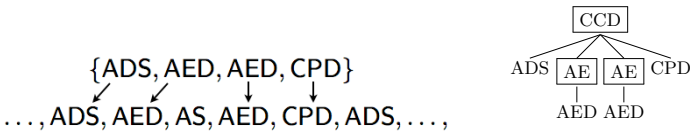


Fig. 4: A Partial Recipe Tree for the Create Correct Device (CCD) Action

To generate all expansions for complex action a, we create a *recipe tree* containing all recipes for a. This structure has two types of nodes: "AND" nodes, whose children represent actions that must be carried out to complete a recipe; and "OR" nodes, whose children represent a choice of recipes for completing an action. The root, action a, is an OR node. For each recipe R_a of a there is an AND child node labeled with the sub-actions of R_a. The children of this AND node are the recipe trees of each sub-action. A partial recipe tree for the CCD action is shown in Figure 4. Dotted leaves denote unfinished sub-trees. The first line of Figure 5a shows the CCD expansion found by selecting the leftmost child at each OR node.

We say that a *match* exists between an interaction and expansion if each action in the expansion can be mapped to a distinct user action with this subset of user actions satisfying all restrictions. Figure 5 shows a match between a



(a) Expansion and Match (b) Plan

Fig. 5: Expansion, Match, and Plan for the CCD Action

CCD expansion and the partial interaction from Figure 2. Note that two user actions, AS and ADS, were deemed redundant based on the match.

Planning is the process by which users use recipes to compose basic and complex actions in order to complete tasks. A *plan* for complex action a is a hierarchical composition of actions such that each complex action is decomposed into sub-actions from a corresponding recipe. The set of basic actions in a plan are an expansion of a. Figure 5b shows one plan for completing the CCD action.

4 From Recipes to Constraint Satisfaction Problems

We now explain how to combine an expansion and user interaction to create a constraint satisfaction problem (CSP). A solution to this CSP gives an explanation of a user's activities by providing a match between the expansion and interaction. A *CSP* is a triple (X, Dom, C). $X = \{x_1, ..., x_n\}$ is a finite set of variables with respective domains $Dom = \{D_1, ..., D_n\}$, each a set of possible values for the corresponding variable, $D_i = \{v_1, ..., v_k\}$, and a set of constraints $\mathcal{C} = \{c_1, ..., c_m\}$ that limit the values that can be assigned to any set of variables.

First, we create variables and domains for our CSP. Let $S = \{s_1, \ldots, s_n\}$ and R be the set of sub-actions and restrictions in the expansion, respectively. Each action in S becomes a unique variable in the CSP.

Each variable's domain is derived from the user's interaction. For each occurrence o_s of action s at index i in the interaction, an element (o_s, i) is added to the domain of s in the CSP. Referring to the interaction of Figures 2 and 5a, the domain of the ADS variable, $Dom(\mathsf{ADS})$, is $\{(\mathsf{ADS}, 0), (\mathsf{ADS}, 5)\}$.

We now create our CSP constraints. For each restriction in R over actions $(s_1 \ldots, s_m) \in S$, we add a constraint over the corresponding CSP variables.

5 Recognition Algorithms

We provide two algorithms that use CSPs to output a plan for complex action a in interaction history h given a database of recipes $\mathcal{R}$. Both algorithms rely on the recipe tree of a to inform their recognition process. The first approach, the Brute-Force algorithm traverses the recipe tree and solves a CSP for every expansion of a or until a match is found. A solution for a CSP provides a match between an expansion and interaction history. The path traversed on the recipe tree to generate that expansion is effectively the user's plan towards completing a task in TinkerPlots.

Use the recipes in $\mathcal{R}$ to construct the recipe tree for complex action a.
Traverse the tree bottom-up. For each OR node representing action s,
  If s has not been cached,
    Use the Brute-Force algorithm to recognize s.
    Cache s as failed or successful.
  If s is cached as failed,
    Prune the parent of s from the tree.
Call the Brute-Force algorithm to recognize a.

Fig. 6: Pruning Recognition Algorithm

The second algorithm is a more sophisticated, bottom-up approach. The Pruning algorithm builds CSPs for sub-actions, pruning nodes from the recipe tree for a if their descendants cannot be explained by the user's interaction. This process narrows the search space of expansions for root action a.

6 Empirical Methodology

We collected interaction histories from 12 adults with backgrounds spanning some high school to some post graduate education. Subject were given identical 30-minute TinkerPlots tutorials and asked to complete four problems in succession. Five of 48 interactions were discarded due to a logging bug causing crucial user actions to be unregistered. Results are based on the remaining 43 instances.

We compared the performance of three recognition algorithms: the Brute-Force and Pruning algorithms and the algorithm proposed by Gal et al. [2], denoted the "Greedy" algorithm. For a plan output by an algorithm to be "correct", a domain expert had to agree on both whether the user solved the problem and which actions played a salient part in the user's solution. [1]

Overall, both CSP algorithms inferred correct solutions for 40 of 43 (93%) interactions, while the Greedy algorithm inferred correct solutions for 27 of 43 (63%) interactions. In all cases, incorrect inferences were "false-negatives," meaning the algorithms were unable to find solutions identified by the domain expert. For the CSP algorithms, all "false-negatives" resulted from limitations of recipe expressiveness, such as the requirement for non-recursive recipes.

For both problems, scalability issues were linked to recipe tree complexity rather than the length of user logs. Correlated to the number of distinct expansions, recipe tree complexity is exponential in both the maximum number of recipes for any action and the maximum number of constituents for any recipe in the recipe database. We find that increased recipe tree complexity for a problem corresponds to increased average run-time. In contrast, the longest user log for each problem experienced among the shortest run-time for that problem, and the average log size for a problem did not correspond to increased average run-time.

User logs ranged in size from 14 to 80 actions, and plans ranged from 16 to 34 actions. The average user log and plan were 35 and 21 actions, respectively. 29 of 43 interactions (70%) contained a solution. Further details and a comparison of both CSP algorithms can be found in a technical report [13].

[1] To implement our algorithms, we used an open-source test-bed by Gustavo Niemeyer for solving the CSPs, available at: http://labix.org/python-constraint.

7 Conclusion and Future Work

This work provided a comprehensive study of the use of constraint satisfaction techniques towards automatic recognition of users' interactions with computer software. Given a comprehensive recipe database, we showed this approach to provide a robust solution. We evaluated our techniques in "real-world" conditions, recognizing users' activities in commercial pedagogical software. These algorithms outperformed a related approach from the literature for inferring user-software interaction. In future work, we wish to evaluate these techniques in other software and to use plans output by our algorithms to construct a collaborative pedagogical agent that generates support to guide student users.

Acknowledgements

This work was supported by NSF grant number REC-0632544. Thanks to Andee Rubin for her great help with the user study and for her paper comments and suggestions. Thanks to Barbara Grosz for her insights and guidance throughout the research and writing processes. Thanks to Elif Yamangil for assisting with the development of the Greedy algorithm, Cliff Konold for helpful discussions, and Craig Miller for developing the TinkerPlots logging facility.

References

1. Pollack, M.: Intelligent technology for an aging population: The use of AI to assist elders with cognitive impairment. AI Magazine **26**(9) (2006)
2. Gal, Y., Yamangil, E., Rubin, A., Shieber, S.M., Grosz, B.J.: Towards collaborative intelligent tutors: Automated recognition of users' strategies. In: Proceedings of Ninth International Conference on Intelligent Tutoring Systems (ITS), Montreal
3. Lesh, N., Rich, C., Sidner, C.: Using Plan Recognition in Human-Computer Collaboration. (1999) 23–32
4. Anderson, J.R., Corbett, A.T., Koedinger, K., Pelletier, R.: Cognitive tutors: Lessons learned. The Journal of Learning Sciences **4(2)** (1995) 167–207
5. Bauer, M.: Acquisition of user preferences for plan recognition. In: Proceedings of the Fifth International Conference on User Modeling. (1996) 105–112
6. Lesh, N.: Adaptive Goal Recognition. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence, Morgan Kaufmann (1997) 1208–1214
7. Conati, C., Gertner, A., VanLehn, K.: Using bayesian networks to manage uncertainty in student modeling. Journal of User Modeling and User-Adapted Interaction **12**(4) (2002) 371–417
8. Corebette, A., McLaughlin, M., Scarpinatto, K.: Modeling student knowledge: Cognitive tutors in high school and college. User Modeling and User-Adapted Interaction **10** (2000) 81—108
9. C. Konold, C.M.: TinkerPlots Dynamic Data Exploration 1.0. Key Curriculum Press. (2004)
10. Hammerman, J.K., Rubin, A.: Strategies for managing statistical complexity with new software tools. Statistics Education Research Journal **3**(2) (2004) 17–41
11. Grosz, B., Kraus, S.: The evolution of sharedplans. Foundations and Theories of Rational Agency (1999) 227–262
12. Pollack, M.: Plans as complex mental attitudes. MIT Press (1990)
13. Reddy, S., Gal, Y., Shieber, S.M.: Recognition of users' activities using constraint satisfaction. Technical report, Harvard University (2009)